

Proposta di un'architettura Weightless per il Deep Learning

Sommario

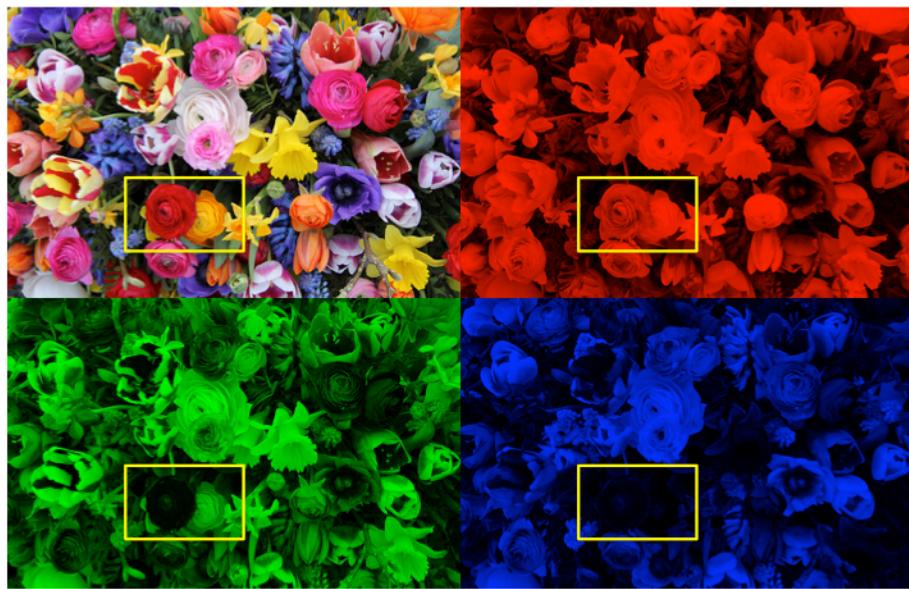
Trasformazione dell'immagine di input a colori in immagini binarie	3
Trasformazione tramite codifica con Minchinton Cell.....	3
Trasformazione tramite codifica con termometri	6
Pooling	7
ReLU.....	8
Filtering	8
Trasformazione in MPLN	10
Dimensionamento delle MPLN	10
Scelta del <i>kernel</i>	13
La rete <i>fully connected</i> (FC).....	13
Dataset da provare (http://deeplearning.net/datasets/)	14
The CIFAR-10 dataset (http://www.cs.utoronto.ca/~kriz/cifar.html)	14
Proposta di architettura MPLN per CIFAR-10	14
The STL-10 dataset (http://ai.stanford.edu/~acoates/stl10/)	15
Proposta di architettura MPLN per STL-10	16
Osservazioni.....	16
Architetture note	17
LeNet 1998.....	17
AlexNet 2012	17
ZfNet 2013	18
VGG 2015.....	19
GoogleNet 2015	19
ResNet 2015.....	20

Inception-V3, V4 and Inception-ResNet 2016	20
WideResNet 2016	21
Xception 2017	21
ResNeXt 2017.....	22
SE-Network 2018	22
Channel Bust CNN using TL 2018	22
RAN 2017	23
CBAM – Convolutional Block Attention Module 2018.....	23
CIFAR	24
CaffeNet	25
Taxonomy of deep CNN architectures showing the seven different categories	25

Trasformazione dell'immagine di input a colori in immagini binarie

Trasformazione tramite codifica con Minchinton Cell

L'input originale RGB formerà 3 immagini diverse una in R una in G e l'altra in B. Quest'ultime possono essere trattate come immagini a tonalità di grigio essendo i valori dei pixel variabili tra 0 e 255.



Trasformazione nei tre canali RGB dell'immagine di input

Come si nota, in ogni canale ci sono dei particolari che negli altri non appaiono (nel riquadro giallo si possono notare le differenze che ci sono nei due fiori).

Se l'immagine d'input ha dimensioni $h \cdot w$, la trasformazione creerà un nuovo "input" di dimensioni $3 \cdot h \cdot w$.

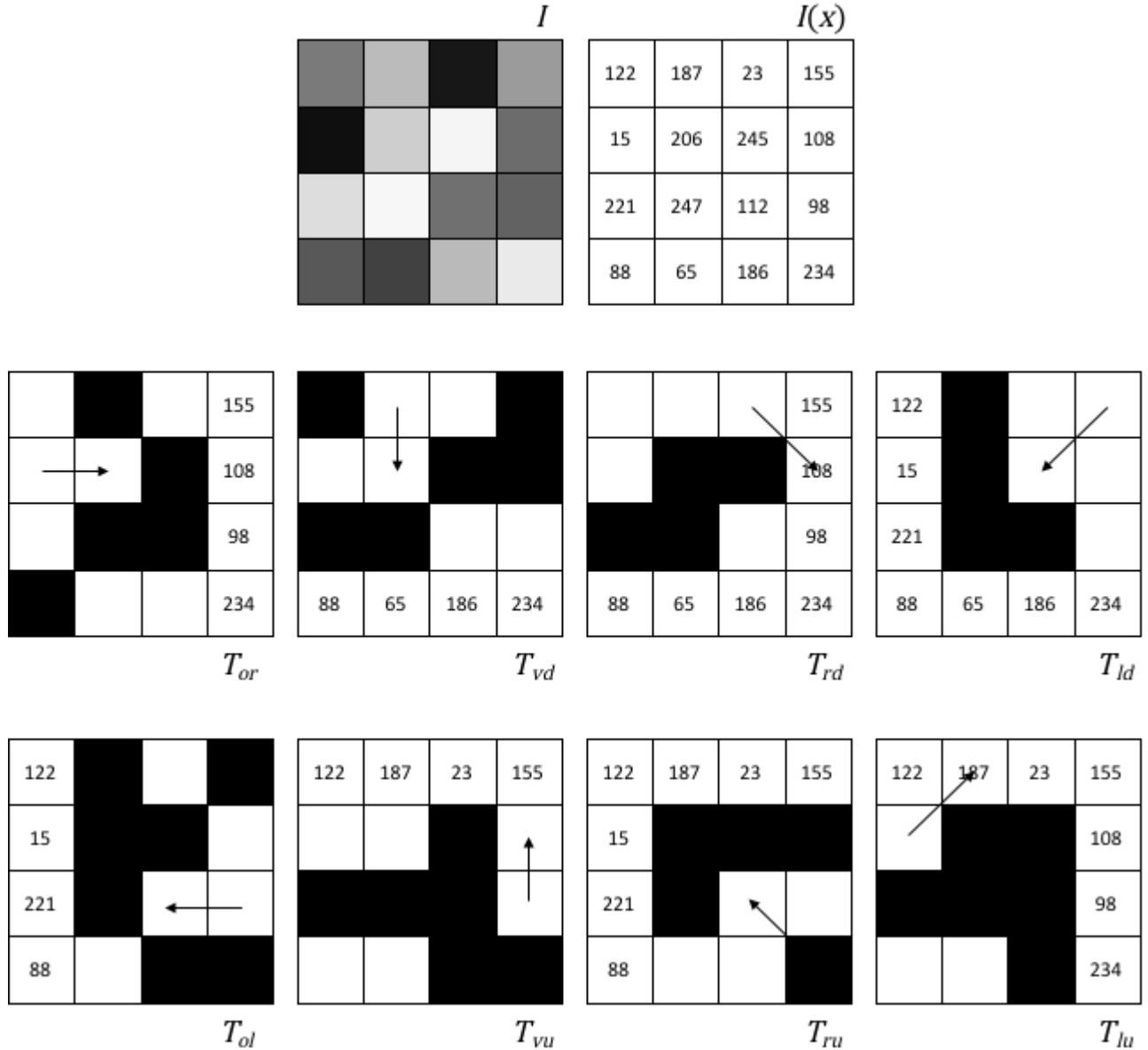
Basandoci sul fatto che quello che l'occhio umano riconosce subito sono le differenze e avendo l'obiettivo di conservare le informazioni dell'immagine anche se l'andiamo a trasformare, applichiamo una trasformazione (codifica) tramite Minchinton Cell (Type 0) a ognuna delle tre immagini:

$$I(x_1) > I(x_2) \Rightarrow T(x_1) = 1$$

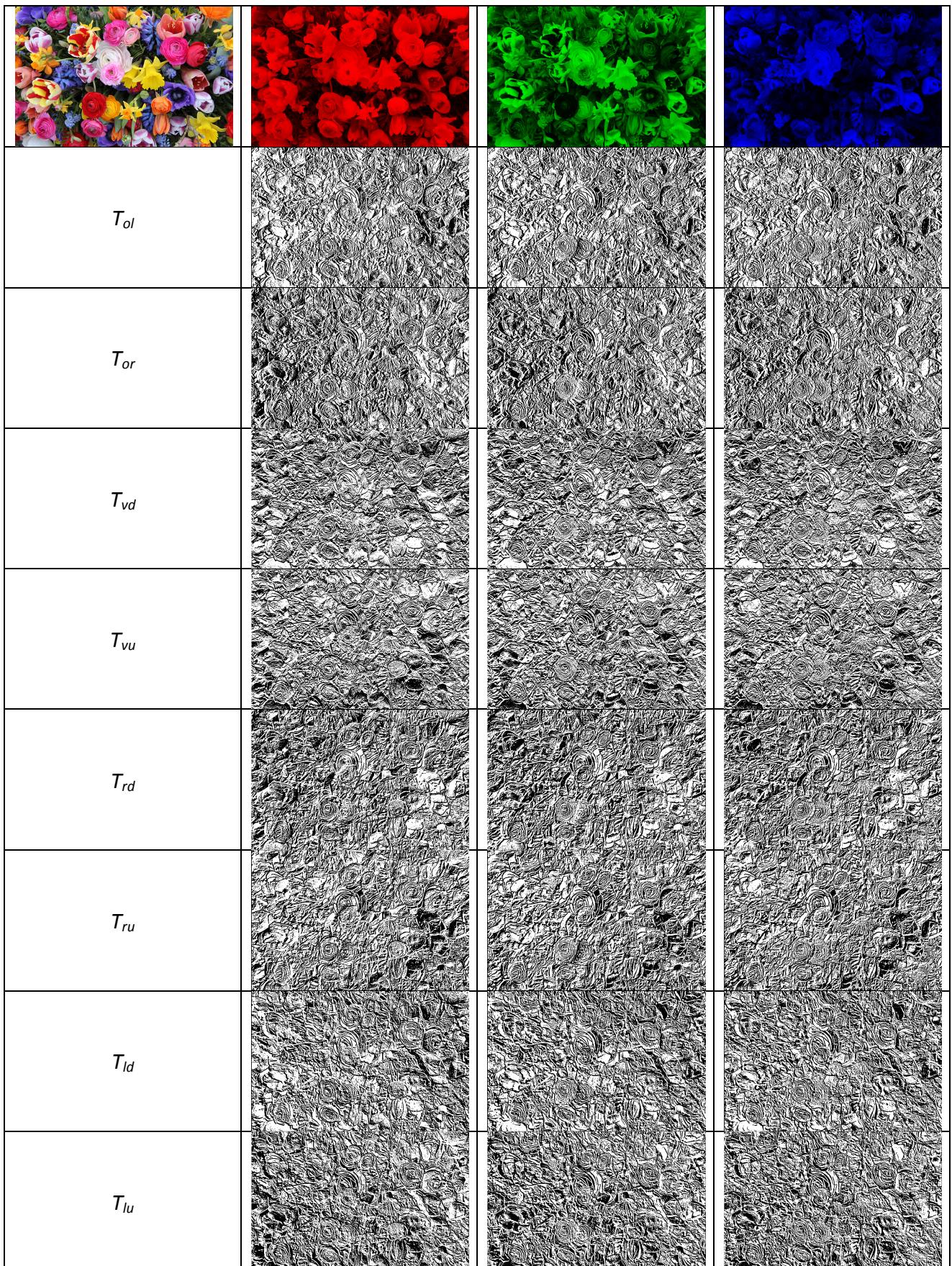
dove $I(x_1)$ rappresenta il colore del pixel di input (I) nella posizione x_1 , $T(x_1)$ il valore che assume il pixel x_1 nell'immagine trasformata (T). Di default tutti i pixel di T sono posti a 255 (bianco) prima della trasformazione.

Queste trasformazioni verranno applicate in orizzontale da sinistra a destra e viceversa (T_{or} e T_{ol}), in verticale dall'alto in basso e viceversa (T_{vd} e T_{vu}) e lungo le diagonali dall'alto in basso e

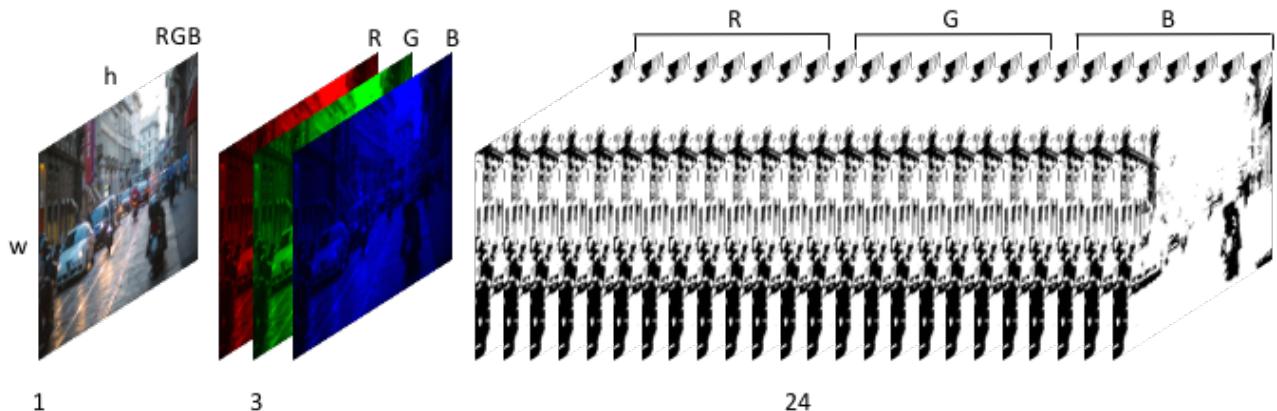
viceversa (T_{rd} e T_{ru} ; T_{ld} e T_{lu}). La loro applicazione comporterà la creazione di 8 immagini con dimensioni diverse da quella di input, cioè: $T_o(x)$ avrà dimensioni $(h-1) \cdot w$, $T_v(x)$ dimensioni $h \cdot (w-1)$, $T_r(x)$ e $T_l(x)$ dimensioni $(h-1) \cdot (w-1)$. Possiamo, comunque, lasciare invariata la dimensioni delle immagini se consideriamo la parte esclusa composta di pixel bianchi.



Il risultato dell'applicazione di queste trasformazioni è riassunto nella tabella seguente.



Il volume che si genera dopo l'applicazione degli otto operatori è schematizzato in figura.

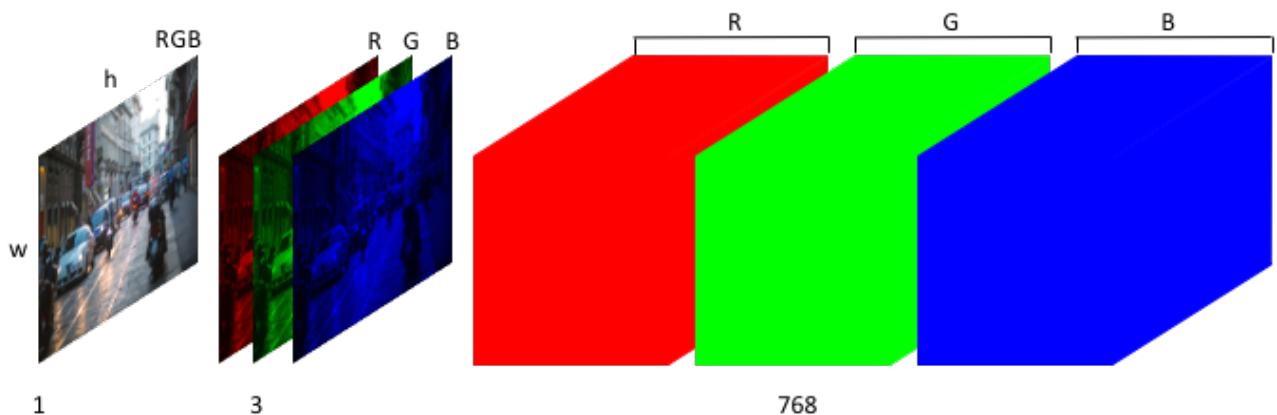


Siamo partiti da un input bidimensionale di dimensioni $h \cdot w$ e abbiamo ottenuto un volume di dimensioni $W=24 \cdot h \cdot w$. Ognuna di queste 24 immagini è binaria e quindi può essere trattata dalle WNN.

Sinceramente mi aspettavo qualcosa di diverso, comunque resta una possibile decodifica in bianco e nero di immagini a colori.

Trasformazione tramite codifica con termometri

Si farà una prova anche con la semplice trasformazione a termometro dei valori dei tre canali. Si parte, quindi, sempre con un'immagine a colori da cui si producono le tre immagini corrispondenti ai canali R, G e B. Il livello di colore di ogni pixel si trasforma in un termometro che varia nell'intervallo [0, 255]. Il risultato di tale operazione è che ognuna delle tre immagini si trasforma in un volume di dimensioni $256 \cdot h \cdot w$. L'input così trasformato è rappresentato da un volume di dimensioni $W=3 \cdot (256 \cdot h \cdot w)$.



A questo punto si pone il problema su quale parte del volume far scorrere i filtri:

1. si considerano tre volumi diversi uno per ogni canale, generando così un unico output per l'intero volume;

2. si considera un unico volume generando un'unica immagine che tiene contro contemporaneamente delle informazioni contenute nei tre canali.

Il considerare i termometri dal basso verso l'alto o da sinistra a destra influenza l'elaborazione del dato? Credo sia opportuno che vengano generati da sinistra verso destra per conservare il significato stesso del termometro, cioè il valore effettivo del canale.

Bisogna comunque pensare che la nostra rete funziona a potenze di b (indirizzamento) e quindi il fatto che siano 3 immagini iniziali (i tre canali RGB e quindi un numero dispari), da cui ricaviamo 3 volumi di altezza 256 (numero pari), otteniamo un volume di input per i filtri che non ci permette di avere un indirizzamento esatto (ci sarà sempre qualche input che non verrà coperto – anche dentro la piramide). Ma questo, come vedremo più avanti, non sarà un problema.

Pooling

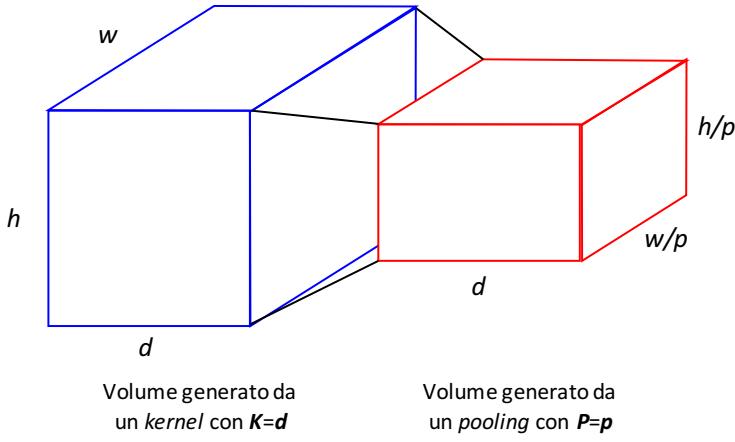
Sia lo stride (S) sia le dimensioni (P) devono essere considerati come parametri. Si potrebbe pensare di far apprendere alla rete anche la strategia di *pooling*. Oppure, una proposta potrebbe essere quella di definire una finestra di *pooling* di $3 \cdot 3$, cioè $P=3$ (o, comunque, sempre di lato dispari) e quindi avere un'uscita 1 se i pixel neri sono maggiori o uguali a 4,5 (cioè $3 \cdot 3 : 2$), 0 viceversa.

Questo perché se consideriamo tutte le possibili combinazioni di pixel neri su un'immagine 3×3 otteniamo le seguenti configurazioni:

N. pixel neri	Configurazioni possibili	Prevalenza colore
0	1	B
1	9	B
2	36	B
3	84	B
4	126	B
5	126	N
6	84	N
7	36	N
8	9	N
9	1	N

quindi con una soglia a 4,5 otteniamo una divisione tra quelli a prevalenza bianca e quelli a prevalenza nera.

Quest'idea nasce dal fatto che siamo costretti a trattare 0 e 1 e quindi operazioni di media, *max* e altro non possono essere applicate. Per fare un'analogia con le immagini, con questa proposta, andiamo a rilevare se quella parte dell'input è più nera o più bianca, che secondo me è l'unica cosa che possiamo fare. Da notare che il *pooling* agisce lungo il volume ma immagine per immagine riducendone la dimensione da $(h \cdot w)$ a $(h/P \cdot w/P)$.



E' da stabilire se lo *stride* debba avere le stesse dimensioni del lato della finestra di *pooling* o se può essere inferiore. Visto che consideriamo lati dispari, lo *stride* potrebbe essere inferiore e quindi generare un'immagine più significativa. Ad esempio, se avessimo un *pooling* $5 \cdot 5$ potremmo adottare uno *stride* di dimensioni 1, 2, 3, e 4 anziché 5.

Anche questo è tutto da provare.

ReLU

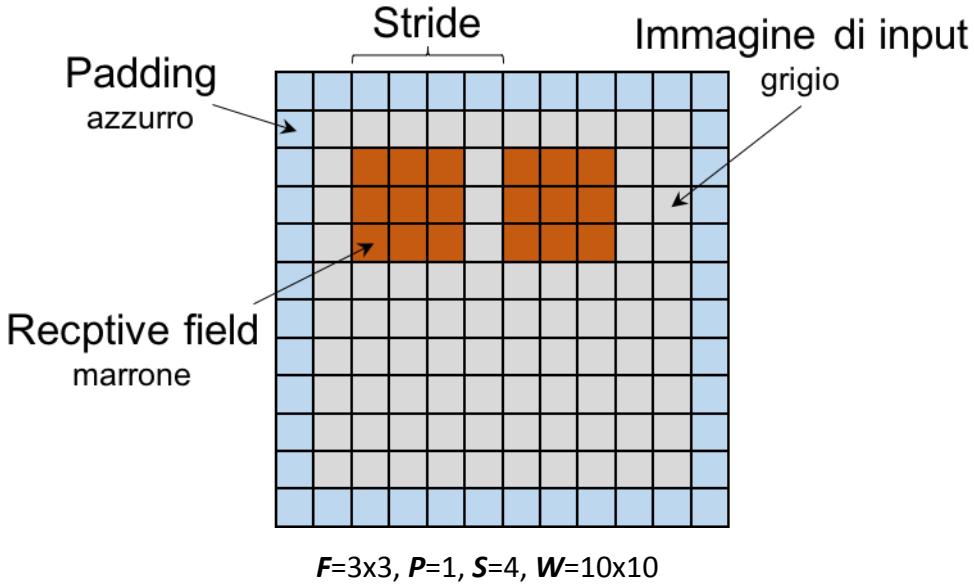
Per le weightless non dovrebbe aver senso: è tutto o 0 o 1 e quindi non c'è la necessità di "raddrizzare" l'input, cioè sostituire col valore zero i valori negativi.

Filtering

La rete organizza, in fase d'addestramento, i suoi propri filtri e per ogni livello convoluzionale i adotta un diverso numero di filtri K_i con *kernel* diversi ma gli stessi per ogni livello.

Leggendo LeNet-5, AlexNet, ... si vede che usano filtri diversi non solo come funzionalità ma anche come dimensione. I filtri saranno realizzati con MPLN a 11 valori (valore consigliato perché è il giusto compromesso tra memoria occupata e *accuracy*).

Anche qui saranno considerati parametri da dare in input: il *receptive field* F (dimensione della finestra $F \cdot F$); dimensione del *padding* P (bordo da aggiungere alle immagini); lo *stride* S (passo di scorrimento del *receptive field*).



Ciò che si evince dalle architetture conosciute e che mi sono andato a spulciare un po', è che il filtro non agisce sulla singola immagine che compone il volume ma direttamente sul volume. Mi spiego. Se la dimensione del volume è $24 \cdot h \cdot w$ il filtro non agisce sulle singole 24 immagini che compongono il volume ma agisce su input di dimensioni $24 \cdot F \cdot F$ cioè lungo tutta la lunghezza del volume.

Il calcolo del numero di pixel (o, equivalentemente, del numero di neuroni dello strato successivo) prodotti dall'applicazione di un singolo filtro è il seguente:

$$Px = \frac{W - F + 2P}{S} + 1.$$

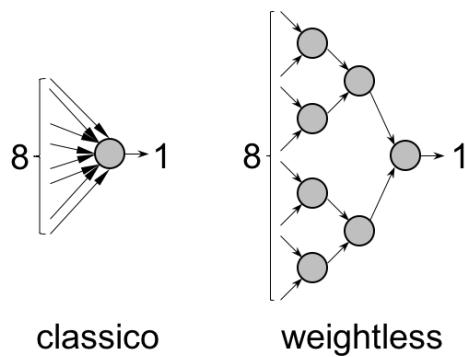
Ad esempio, per un input $W=7 \cdot 7$, un filtro $F=3 \cdot 3$ con $S=1$ e $P=0$ si ottiene un output (o un'immagine d'uscita) di dimensioni $5 \cdot 5$. Da notare che questo viene fatto per ogni dimensione dell'immagine, cioè il calcolo prevede che si inserisca, per l'esempio riportato, $W=7$ e $F=3$.

Credo che la dimensione del filtro debba essere sempre dispari per poter associare a un pixel il suo esatto intorno. Con una dimensione pari ciò non è possibile.

Sempre riferendoci all'esempio di prima, è importante notare che ogni filtro scorre l'intera area $h \cdot w$ del volume di input $24 \cdot h \cdot w$. Quindi lo stesso filtro è applicato a tutti i pixel che compongono l'area di base del volume. Ad esempio, se abbiamo un volume di dimensioni $256 \cdot 64 \cdot 64$ con area di base $64 \cdot 64$, il generico filtro K_i sarà applicato a tutti i $64 \cdot 64$ pixel della base del volume (ovviamente dipende dallo *stride* e dal *padding*) ricevendo in input un volume $256 \cdot F \cdot F$ per $64 \cdot 64$ volte. In questo modo si forma una singola immagine che andrà a comporre il volume di uscita. Quest'ultimo avrà profondità uguale al numero di filtri che vogliamo applicare.

Trasformazione in MPLN

Consideriamo un filtro di dimensioni 3·3 applicato al volume di input che ha una profondità di 3·256. Abbiamo un totale di 6912 pixel che formano l'input del filtro. Nelle CNN si utilizza un singolo neurone con un *fan-in* uguale a 6912 e un *fan-out* uguale a 1. Gli output di questi neuroni formano la nuova immagine dello strato successivo che di solito, ma non è detto, è un livello di *pooling*. L'equivalente WNN si ottiene se sostituiamo il neurone classico con una piramide di MPLN che ovviamente avrà gli stessi *fan-in* e *fan-out* dell'equivalente classico. Anche in questo caso, le uscite delle piramidi formeranno l'immagine che poi sarà data in pasto al *layer* successivo (*pooling* o convoluzionale).



L'esempio riportato in figura è molto riduttivo ma serve a far capire la trasformazione da classico a weightless: 2 bit appiattiscono troppo l'informazione.

Dimensionamento delle MPLN

Supponiamo di utilizzare un indirizzamento fisso per ogni *layer* della piramide di MPLN.

L'esempio riportato in Fig. 1 si riferisce a un volume di input pari a: $256 \cdot 3 \cdot 3 \cdot 3$, dove 256 è il termometro associato a un canale, 3 è il numero di canali e 3·3 la dimensione del filtro. Il numero di pixel da dare in input alla rete è 6912. Con un indirizzamento $b=3$, si ottengono nel primo *layer* 2304 neuroni e nessun input scoperto. Ma nei *layer* 4, 5 e 6 rimane scoperto 1 input.

Layer	N. neuroni	Input
1	2304	0
2	768	0
3	256	0
4	85	1
5	28	1
6	9	1
7	3	0
8	1	0

Fig. 1

In Fig. 2, invece, si nota che con $F=5$ e $b=5$ (cioè con un input di dimensioni 19200), ci sono più output che non vengono coperti dal *layer* successivo.

<i>Layer</i>	N. neuroni	Input
1	3840	0
2	768	0
3	153	3
4	30	3
5	6	0
6	1	1

Fig. 2

Tre sono le possibili opzioni:

1. lasciamo tutto come sta trascurandoli;
2. aggiungiamo nuovi neuroni al *layer* successivo;
3. adottiamo indirizzamenti diversi per ogni layer.

Nel primo caso, l'informazione del singolo canale di output di un neurone intermedio contiene molta informazione che verrebbe persa.

Nel secondo caso e riferendoci all'esempio di fig. 2, nel terzo *layer* possiamo aggiungere un nuovo neurone che assorbe i 3 input scoperti e gli altri due li possiamo prendere casualmente dagli altri. Ma facendo due conti non solo sarebbe difficile implementare il meccanismo, ma si distruggerebbero in cascata i livelli successivi.

Il terzo caso sembra essere l'opzione più appropriata. Nella tabella 1 nella pagina seguente sono riportate tutte le configurazioni interessanti in relazione alla dimensione del *receptive field*. Come si nota, ogni livello ha un suo indirizzamento. Quest'organizzazione permette di avere delle piramidi di MPLN senza perdita di informazione come nei casi 1 e 2. Il problema però adesso si sposta su come scegliere la configurazione più adatta rispetto alla dimensione del filtro. L'idea che mi sono fatto è che non dovremmo avere neuroni con indirizzamenti troppo bassi (2, 3 e 4) perché ci sarebbe un appiattimento delle informazioni con relativo degrado delle performance.

Nella tabella 1 e per la rappresentazione termometro, sono state evidenziate in rosso quelle che ritengo siano le più interessanti dal punto di vista sia della quantità di informazione che possono gestire, sia delle performance dell'intero sistema.

C'è da fare un ragionamento su come scegliere l'indirizzamento dei *layer*. Se nei primi *layer* adottiamo un indirizzamento alto, questi sono predisposti ad accettare input di grandi dimensioni e molto variabili che è una caratteristica tipica dei *dataset* che andremo a trattare. Inoltre, sarebbero molto sensibili alle variazioni degli input e ci permetterebbero di non saturare le RAM.

Lo stesso ragionamento va fatto per gli ultimi *layer*. Indirizzamenti piccoli tenderebbero non solo a far saturare velocemente i nodi, ma non avrebbero quella sensibilità che è richiesta per questi sistemi. Ricordiamo che: indirizzamenti piccoli sono molto precisi ma poco sensibili mentre indirizzamenti grandi sono molto sensibili a piccole variazioni e quindi poco precisi.

Questi sono i motivi per cui ho scelto determinate configurazioni nella tabella e ne ho escluso altre. Dobbiamo decidere come regolarci.

Non sono andato oltre il 9·9, ma non è difficile generarli.

Volume 6912	Config.	layer	1	2	3	4	5	N. neuroni	Loc. mem.
Filtro 3x3	1	bit	16	12	6	6			
		N. neuroni	432	36	6	1		475	28459456
	2	bit	12	12	8	6			
		N. neuroni	576	48	6	1		631	2557504
	3	bit	12	9	8	8			
Filtro 5x5		N. neuroni	576	64	8	1		649	2394368
	4	bit	8	8	6	6	3		
		N. neuroni	864	108	18	3	1	994	250184
	5	bit	8	6	6	6	4		
		N. neuroni	864	144	24	4	1	1037	232208
Volume 19200	Config.	layer	1	2	3	4	5	N. neuroni	Loc. mem.
Filtro 5x5	1	bit	16	15	10	8			
		N. neuroni	1200	80	8	1		1289	81273088
	2	bit	16	12	10	10			
		N. neuroni	1200	100	10	1		1311	79064064
	3	bit	10	10	8	6	4		
Filtro 7x7		N. neuroni	1920	192	24	4	1	2141	2169104
	4	bit	10	10	8	8	3		
		N. neuroni	1920	192	24	3	1	2140	2169608
	5	bit	10	8	8	6	5		
		N. neuroni	1920	240	30	5	1	2196	2035552
Volume 37632	Config.	layer	1	2	3	4	5	N. neuroni	Loc. mem.
Filtro 7x7	1	bit	16	12	7	7	4		
		N. neuroni	2352	196	28	4	1	2581	154947600
	2	bit	14	14	12	4	4		
		N. neuroni	2688	192	16	4	1	2901	47251536
	3	bit	14	14	8	6	4		
Filtro 9x9		N. neuroni	2688	192	24	4	1	2909	47192336
	4	bit	14	12	8	7	4		
		N. neuroni	2688	224	28	4	1	2945	44965392
	5	bit	12	8	8	7	7		
		N. neuroni	3136	392	49	7	1	3585	12958976
Volume 62208	Config.	layer	1	2	3	4	5	N. neuroni	Loc. mem.
Filtro 9x9	1	bit	16	12	9	6	6		
		N. neuroni	3888	324	36	6	1	4255	256149952
	2	bit	12	12	12	6	6		
Filtro 9x9		N. neuroni	5184	432	36	6	1	5659	23151040
	3	bit	12	9	9	8	8		
		N. neuroni	5184	576	64	8	1	5833	21563648

Tabella 1 – rappresentazione termometro

Un modo per automatizzare la generazione dei *layer* di MPLN sia nella parte convoluzionale sia in quella *fully connected*, potrebbe essere il seguente. Per poter dimensionare i *layer* di MPLN è opportuno scomporre in fattori primi la dimensione dell'input e accoppiare i fattori in modo da ottenere indirizzi che:

1. non abbiamo un valore minore o uguale a 4;
2. non abbiamo un valore troppo grande (non vorrei superare i 16 bit).

Quindi, il primo passo è scomporre in fattori primi la dimensione dell'input. Dai fattori ottenuti si passa al loro raggruppamento in modo da rispettare i punti 1. e 2. Le combinazioni così ottenute si ordinano dall'indirizzamento più alto (primo *layer*) a quello più basso (ultimo *layer*), ovviamente se decidiamo di usare questa strategia. Tra le combinazioni così ottenute, dobbiamo decidere come scegliere quella da adottare.

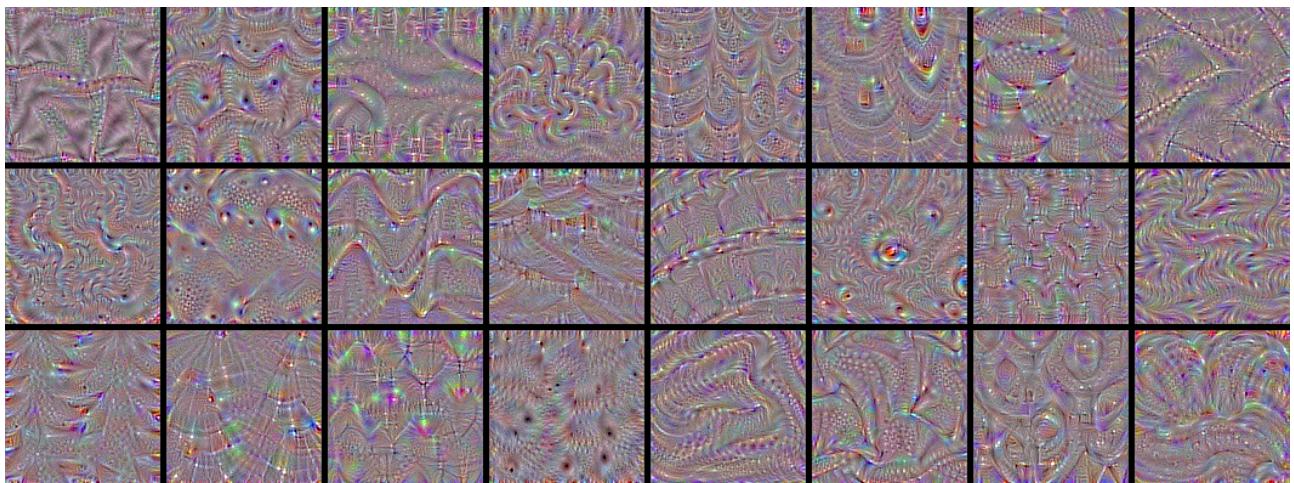
Scelta del *kernel*

Oltre alla dimensione del *receptive field*, si ha la necessità di definire, livello per livello, il *kernel*: il numero di filtri da utilizzare e le loro dimensioni. Ovviamente, non ho la più pallida idea su cosa si basi la scelta di questo numero. Credo faccia parte del *tuning* del sistema.

La rete *fully connected* (FC)

Qui dobbiamo progettare un'architettura in cui tutti i neuroni di ogni *layer* abbiano come *fan-in* tutto l'input del *layer* precedente. L'ultimo layer, inoltre, sarà formato da tanti neuroni quante sono le classi su cui la rete dovrà lavorare.

La prima cosa da decidere è quale tipo di input vogliamo dare a questa rete. Come riportato in alcuni video su youtube, è comodo e interessante vedere l'input formato dalle immagini dell'ultimo strato di filtraggio. Ad esempio potremmo avere 32 immagini (output della parte CNN) da dare in pasto contemporaneamente a tutti i neuroni del primo *layer* della rete *fully connected*.



Esempio: 24 filtri di Keras

La scelta del dimensionamento delle piramidi MPLN che formeranno i layer *FC* sarà fatta in base alla dimensione dell'input e applicando la procedura dei fattori primi.

Quello che bisogna decidere è il numero di *layer* che vogliamo ricordandoci sempre che l'ultimo *layer* sarà composto da tante piramidi MPLN quante sono le classi su cui la rete dovrà lavorare.

Dataset da provare (<http://deeplearning.net/datasets/>)

Di seguito vorrei provare a riportare delle possibili architetture MPLN per *dataset* noti. Si potrebbero poi confrontare i risultati con quelli raccolti in

http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html.

The CIFAR-10 dataset (<http://www.cs.utoronto.ca/~kriz/cifar.html>)

Caratteristiche:

- 60000 immagini 32·32 a colori
- 10 classi con 6000 immagini per classe
- classi: aerei, auto, uccelli, gatti, cervi, cani, rane, cavalli, navi, camion
- le classi sono mutualmente esclusive, non ci sono sovrapposizioni
- 50000 immagini per il *training set*
- 10000 immagini per il test

The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.

Proposta di architettura MPLN per CIFAR-10

La prima proposta è quella di organizzare una rete senza *pooling* (il *pooling* è di solito utilizzato per ridurre le dimensioni dell'input per lo strato successivo). La rete sarà composta da tre convoluzioni (C1, C2 e C3) e da una rete FC formata di 3 *layer* (FC1, FC2 e FC3). L'input iniziale è un'immagine 32·32 a colori, quindi la nostra rete dovrà gestire un volume di input pari a $32 \cdot 32 \cdot 3 \cdot 256 = 786432$ pixel bianchi e neri. Nelle tabelle 1 e 2 sono riportate le caratteristiche per due reti che hanno rispettivamente $K=(64, 32, 16)$ e $K=(128, 64, 32)$.

$F=3, P=1, S=1$	C1 – $K=64$	C2 – $K=32$	C3 – $K=16$	FC1	FC2	FC3
Input	6912 ($3 \cdot 3 \cdot 3 \cdot 256$)	576 ($3 \cdot 3 \cdot 64$)	288 ($3 \cdot 3 \cdot 32$)	16384 ($32 \cdot 32 \cdot 16$)	512	64
<i>Layers (bit)</i>	12 – 9 – 8 – 8	9 – 8 – 8	8 – 6 – 6	16 – 16 – 8 – 8	8 – 8 – 8	8 – 8
Piramidi	64	32	16	512	64	10
Output	65536	32768	16384	512	64	10 classi
N. RAM	649	73	43	561664	4672	90
Memoria	1.89 MB	34.6 KB	9.4 KB	32 GB	1.1 MB	22.5 KB

Tabella 1 - $K=(64, 32, 16)$

$F=3, P=1, S=1$	C1 – $K=128$	C2 – $K=64$	C3 – $K=32$	FC1	FC2	FC3
Input	6912 ($3 \cdot 3 \cdot 3 \cdot 256$)	1152 ($3 \cdot 3 \cdot 128$)	576 ($3 \cdot 3 \cdot 64$)	32768 ($32 \cdot 32 \cdot 32$)	512	64
<i>Layers (bit)</i>	12 – 9 – 8 – 8	16 – 9 – 8	9 – 8 – 8	16 – 16 – 16 – 8 (o 5 da 8)	8 – 8 – 8	8 – 8
Piramidi	128	64	32	512	64	10
Output	131072	65536	32768	512	64	10 classi

Tabella 2 - $K=(128, 64, 32)$

(Non ho fatto il calcolo di memoria e piramidi per il secondo caso).

Forse 512 neuroni (piramidi MPLN) nello strato **FC1** sono troppi (32 GB di memoria). In base agli indirizzamenti dello strato **FC2**, ci possiamo regolare quanti neuroni devono comporre lo strato **FC1**. Ad esempio, se per lo strato **FC2** scegliamo piramidi con indirizzamento 9, 7 e 6, in **FC1** ci vogliono 378 neuroni (piramidi) e non 512. Anche questo si deve decidere.

Credo che per questo tipo di problema non ci interessa avere strati di *pooling* (anche perché l'input non è eccessivamente grande). Ricordiamo che ogni strato di *pooling* riduce l'input per il *layer* successivo. Nel nostro caso, i *pooling* sono sempre dispari e perderemmo l'informazione dei pixel sul bordo: non c'è modo con una finestra dispari non escludere i pixel sul bordo a meno che non li si voglia eliminare dalla computazione. Mi spiego. Se noi centriamo la nostra finestra di *pooling* su un pixel degli angoli avremmo 4 pixel che rientrano nella computazione e 5 che verrebbero esclusi (con il nostro *pooling* non ci possiamo permettere di metterli tutti o bianchi o neri), ottenendo una valutazione di un numero pari di pixel. Nel caso di un pixel sui bordi, avremmo 6 pixel (quindi pari) nella computazione e 3 esclusi ricadendo nel caso precedente.

The STL-10 dataset (<http://ai.stanford.edu/~acoates/stl10/>)

The STL-10 dataset is an image recognition dataset for developing unsupervised feature learning, deep learning, self-taught learning algorithms. It is inspired by the CIFAR-10 dataset but with some modifications. In particular, each class has fewer labeled training examples than in CIFAR-10, but a very large set of unlabeled examples is provided to learn image models prior to supervised training. The primary challenge is to make use of the unlabeled data (which comes from a similar but different distribution from the labeled data) to build a useful prior. We also expect that the higher resolution of this dataset (96x96) will make it a challenging benchmark for developing more scalable unsupervised learning methods.

Overview

- 10 classes: airplane, bird, car, cat, deer, dog, horse, monkey, ship, truck.
- Images are 96x96 pixels, color.
- 500 training images (10 pre-defined folds), 800 test images per class.

- 100000 unlabeled images for unsupervised learning. These examples are extracted from a similar but broader distribution of images. For instance, it contains other types of animals (bears, rabbits, etc.) and vehicles (trains, buses, etc.) in addition to the ones in the labeled set.
- Images were acquired from labeled examples on ImageNet.

Proposta di architettura MPLN per STL-10

Qui dobbiamo trattare sempre immagini a colori ma $92 \cdot 92$ e quindi un volume pari a $92 \cdot 92 \cdot 3 \cdot 256 = 6500352$ pixel (o voxel che è più corretto). Possiamo adottare la stessa architettura di CIFAR-10 con 3 strati convoluzionali e 3 strati FC.

Per ridurre l'input, si può sperimentare un *pooling* binario $3 \cdot 3$ con la funzione introdotta prima, cioè di soglia a 4,5. Adottando una finestra di *pooling* $3 \cdot 3$, uno *stride* 3 e un *padding* -1 (cioè leviamo i pixel dei bordi), l'input per **C2** si ridurrebbe a $30 \cdot 30 \cdot 64$ voxel. Non so quanto possa convenire.

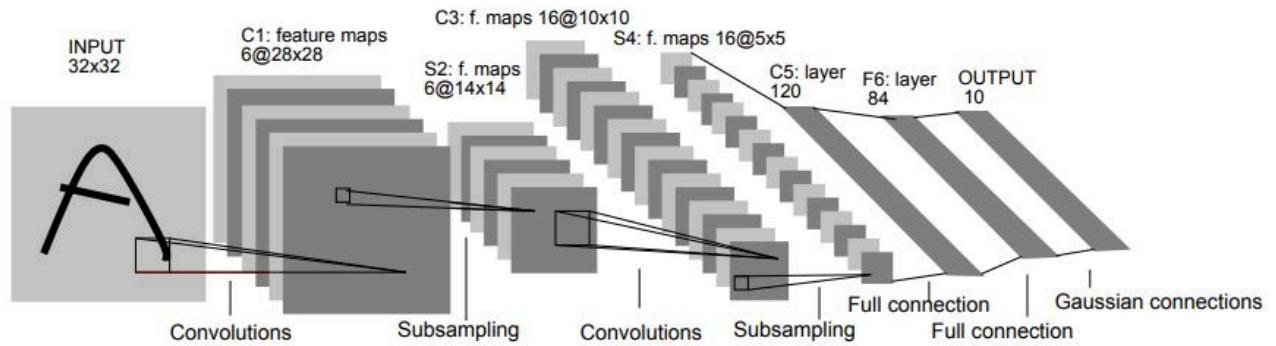
Osservazioni

Non riesco a capire perché le architetture riportate di seguito (in particolare LeNet, AlexNet, ZfNet, VGG 2015, Xception 2017, CaffeNet) adottano sempre un numero crescente di filtri man mano che scendono nella convoluzione. Io ho organizzato l'architettura esattamente al contrario, prima più filtri e poi a decrescere.

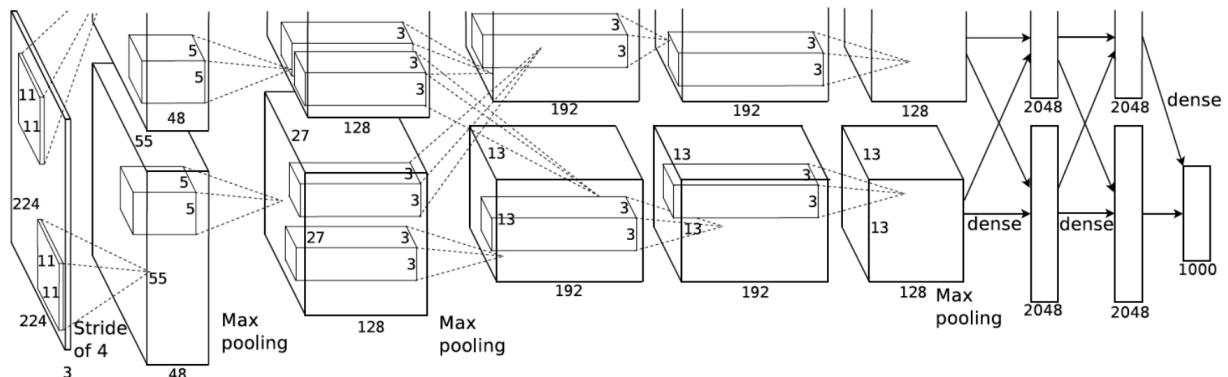
Mi sembra più plausibile raccogliere quante più *feature* dall'immagine originale (più filtri), per poi raffinarle di volta in volta (meno filtri) per creare delle *meta-feature* da quelle di base. Quest'ultimi mettono insieme informazioni di basso livello per crearne di alto livello, quindi dovrebbero essere molti filtri all'inizio e diminuire man mano che si scende nella convoluzione.

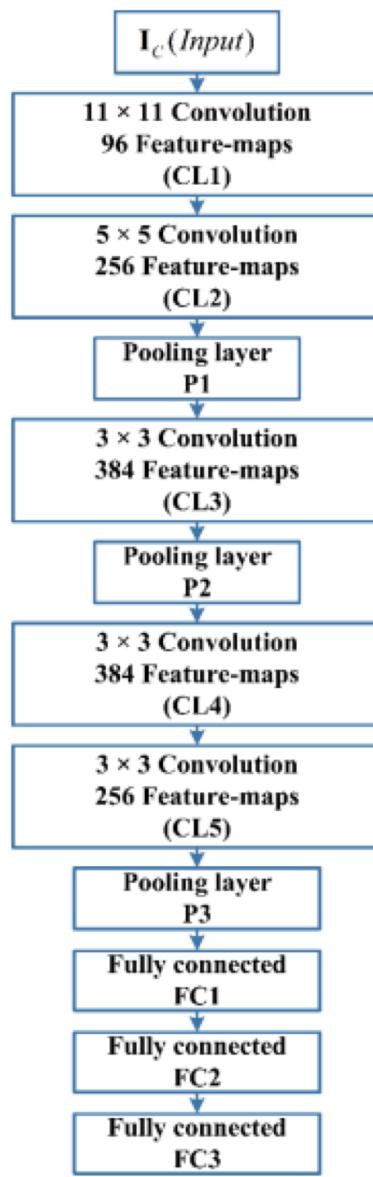
Architetture note

LeNet 1998

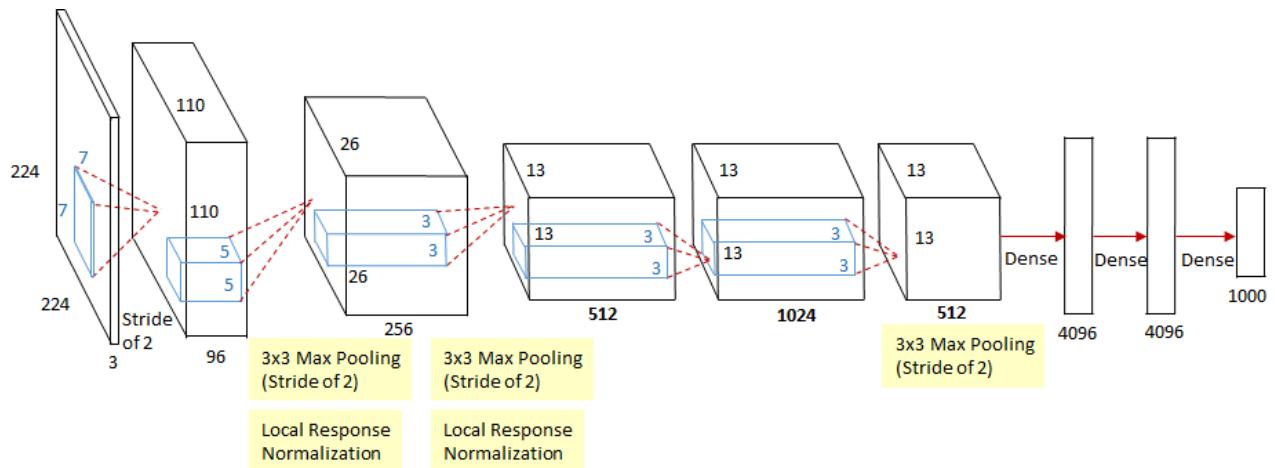


AlexNet 2012

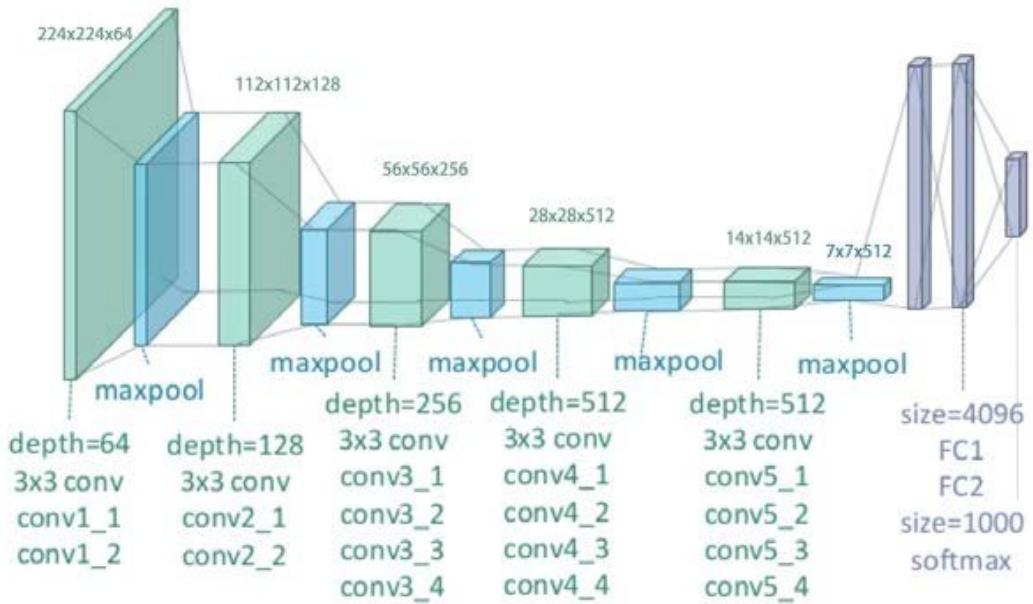




ZfNet 2013



VGG 2015



GoogleNet 2015

The winner of the 2014-ILSVRC competition and is also known as Inception-V1.

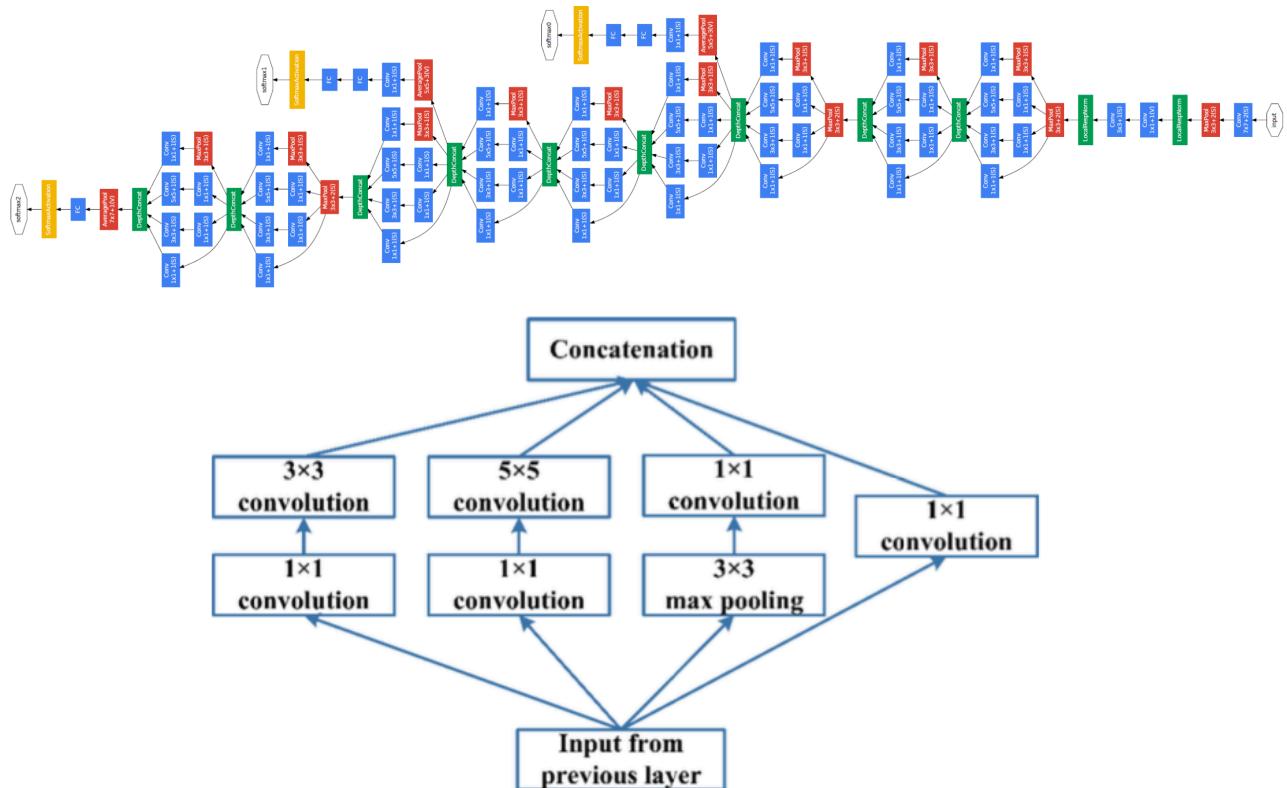
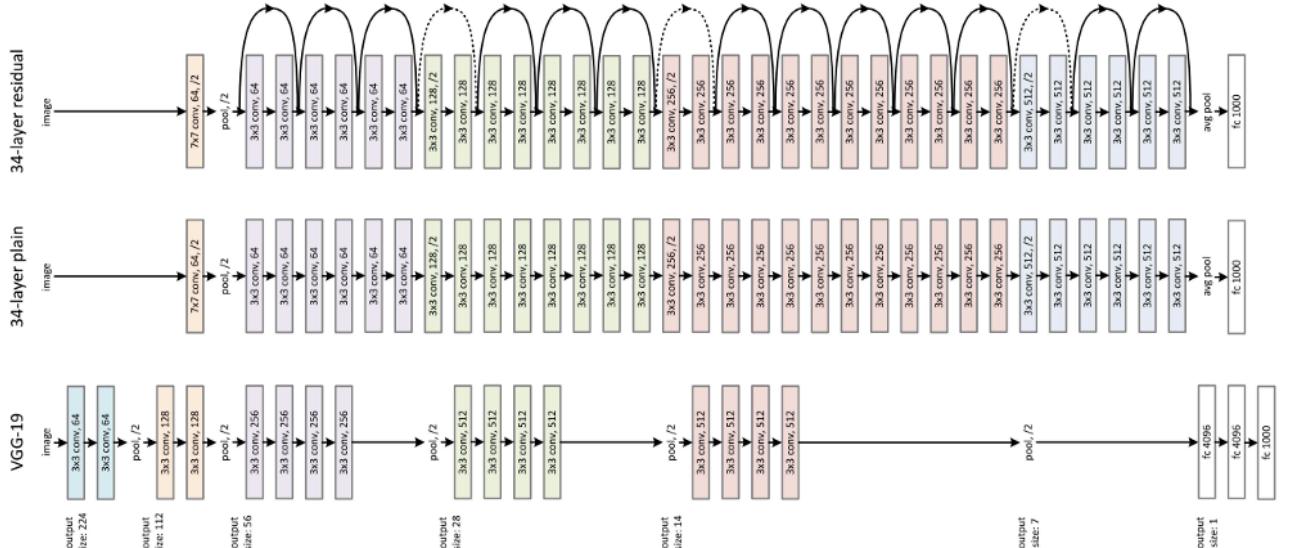
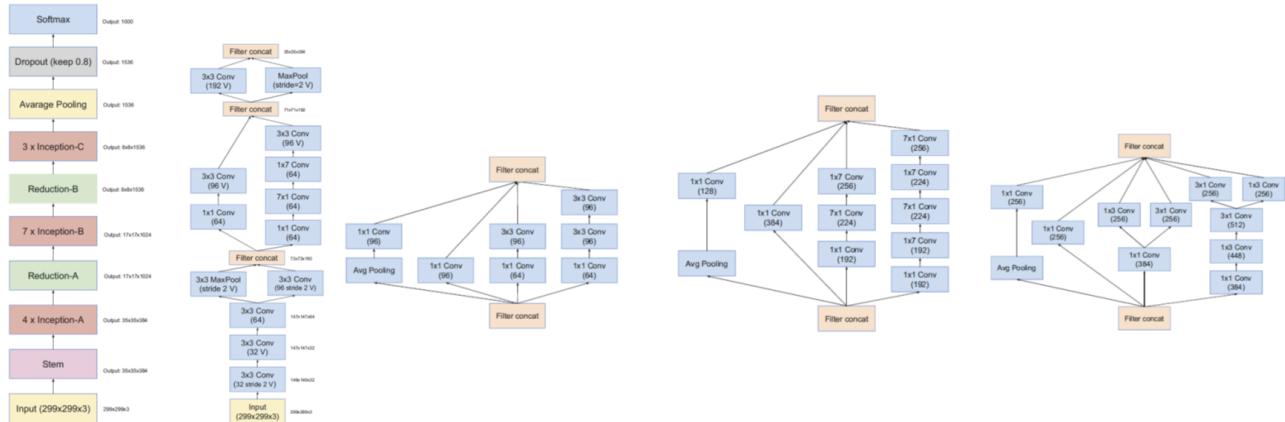


Fig. 6 Basic architecture of the inception block showing the split, transform and merge concept.

ResNet 2015



Inception-V3, V4 and Inception-ResNet 2016



WideResNet 2016

Xception 2017

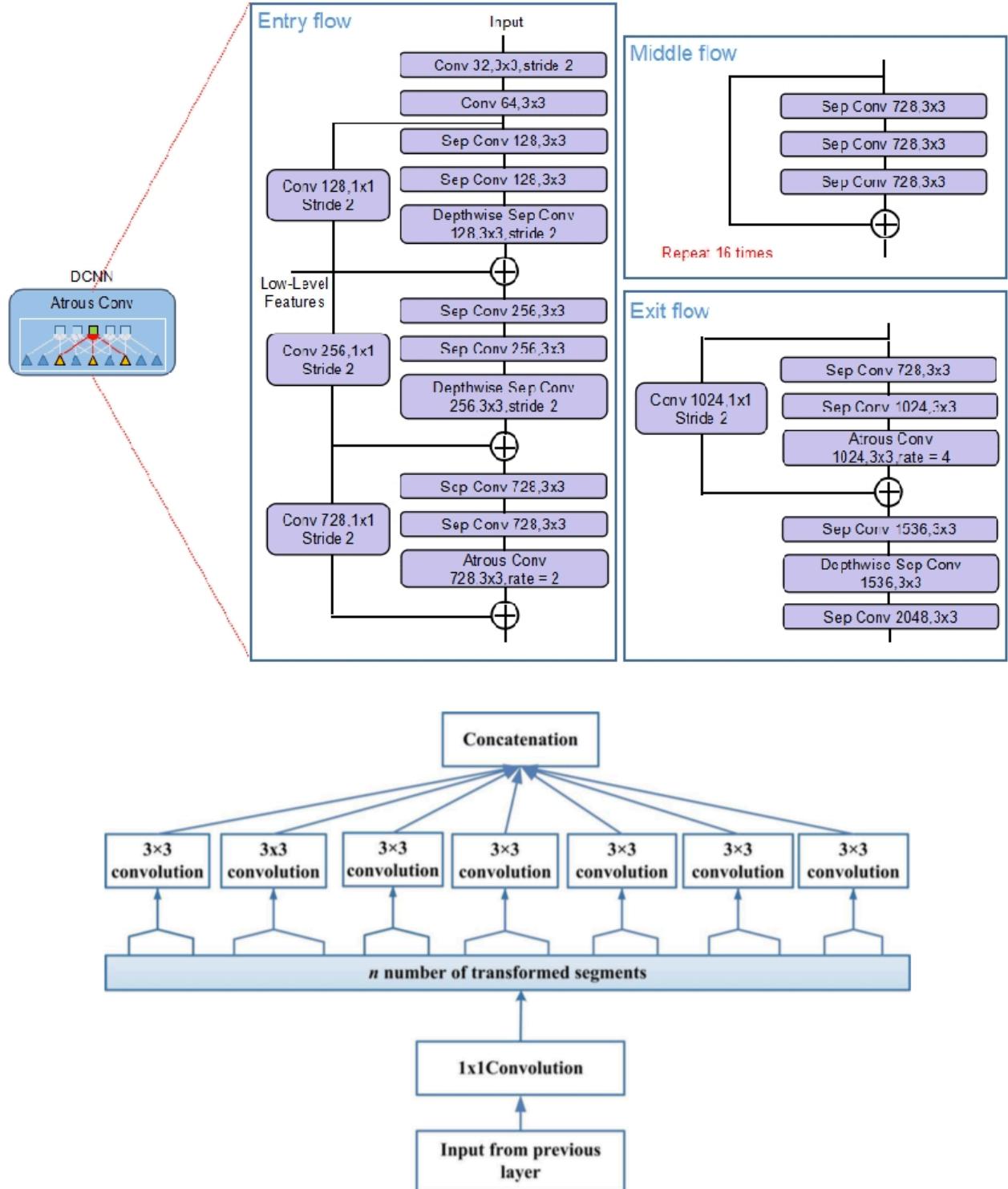
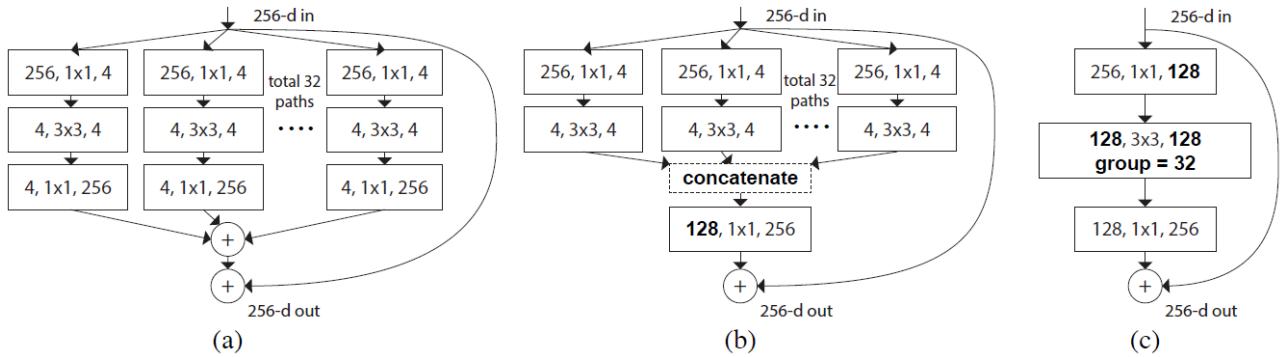


Fig. 8 Xception building block and its n sets of transformation.

ResNeXt 2017



SE-Network 2018

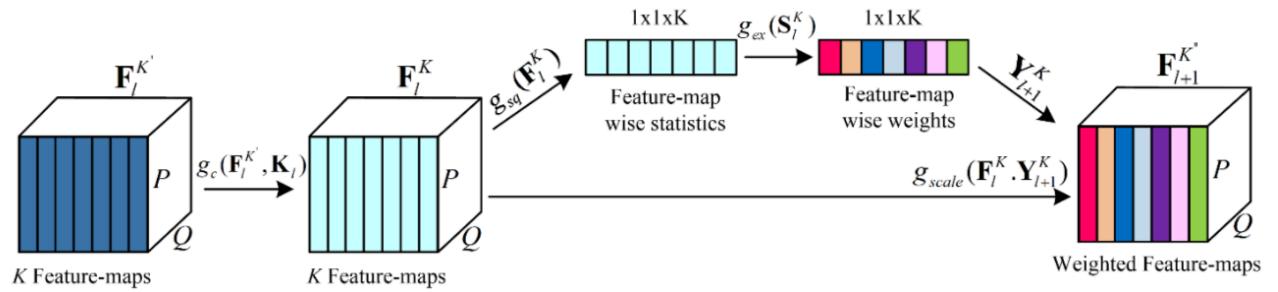


Fig. 10 Squeeze and Excitation block showing the computation of masks for the recalibration of feature-maps that are commonly known as channels in literature.

Channel Bust CNN using TL 2018

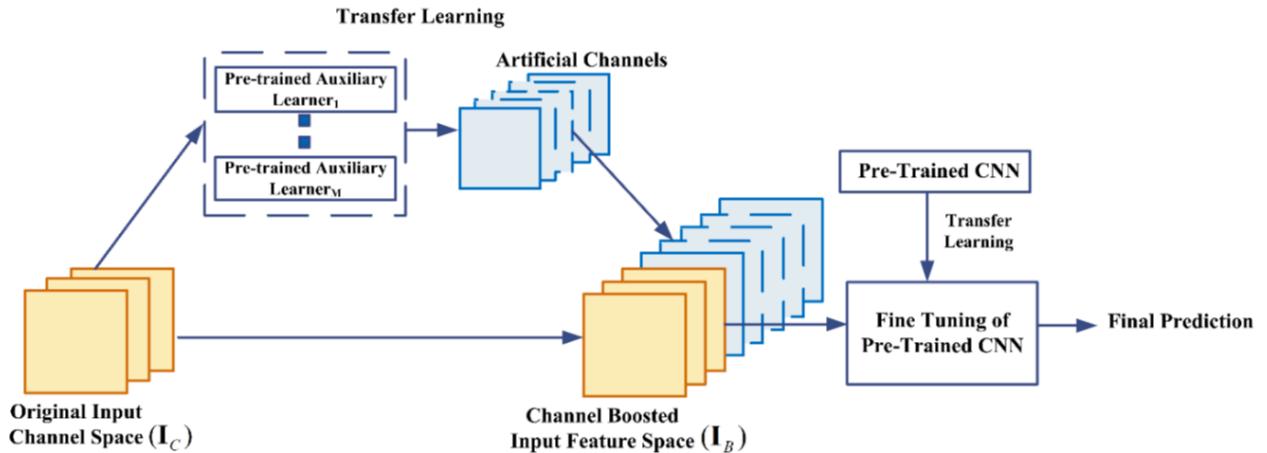
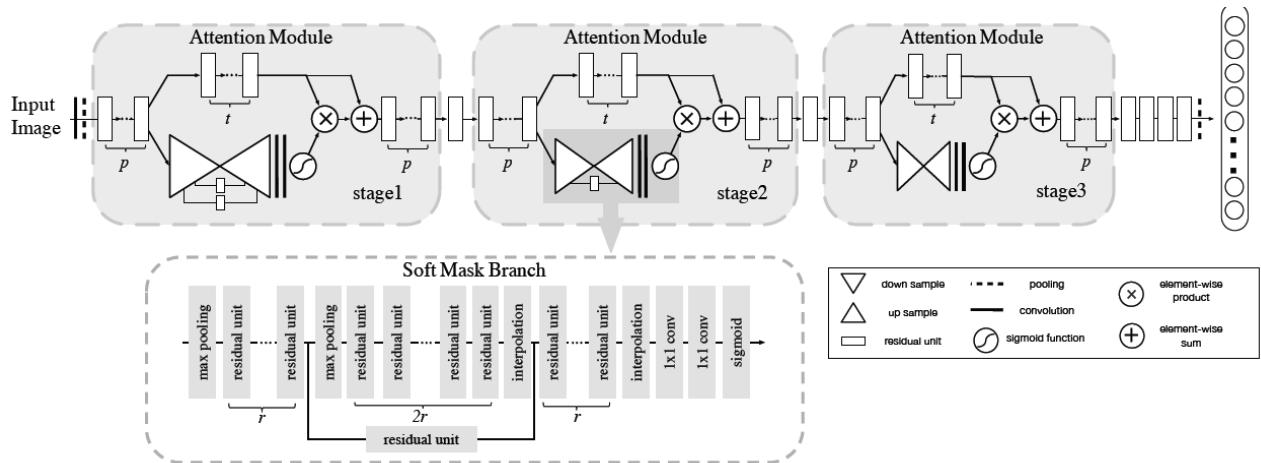
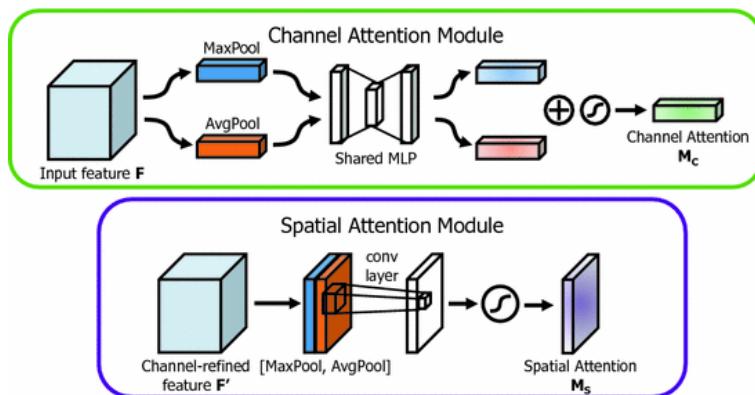


Fig. 11 Basic architecture of CB-CNN showing the deep auxiliary learners for creating artificial channels.

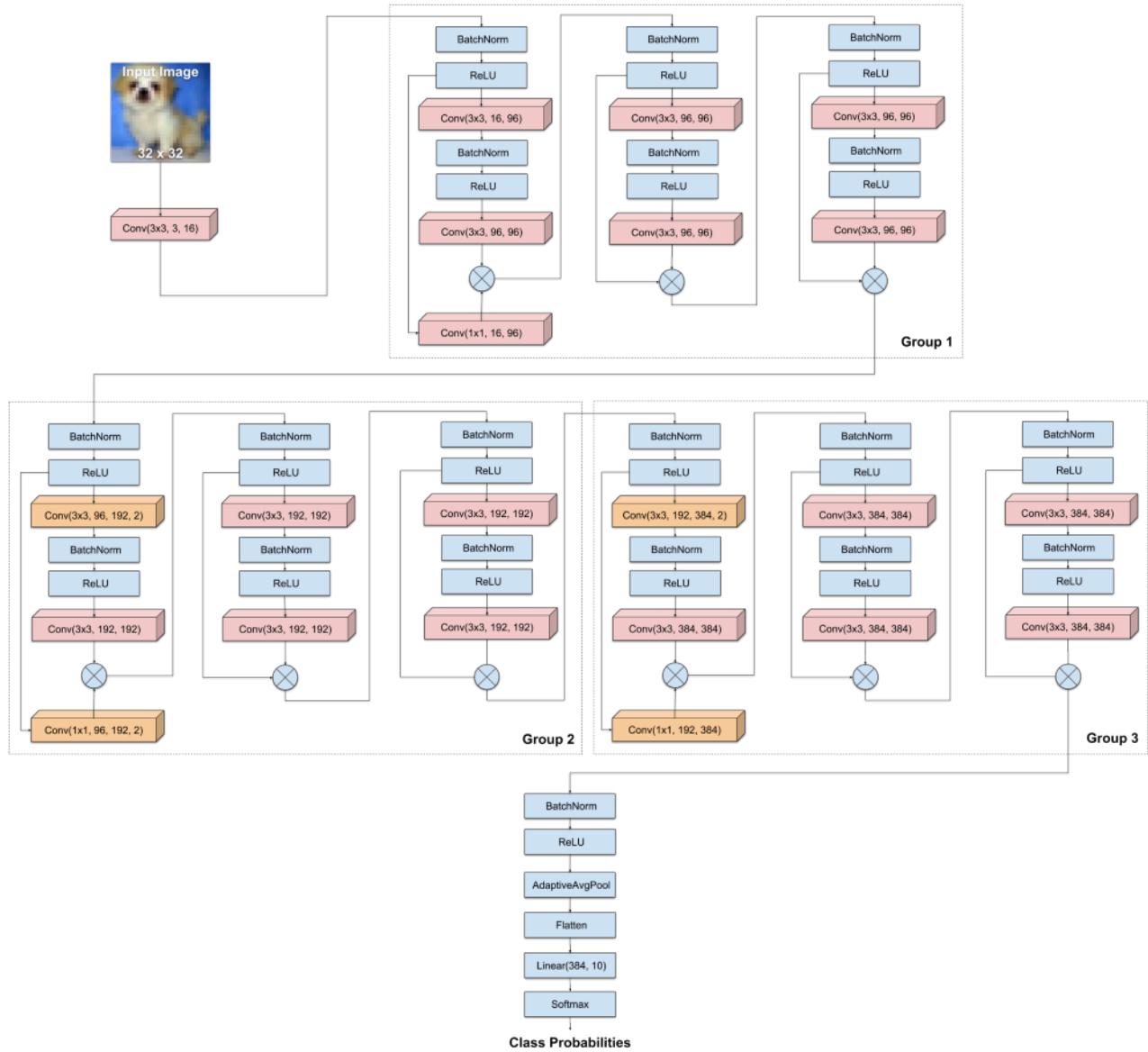
RAN 2017



CBAM – Convolutional Block Attention Module 2018



CIFAR



CaffeNet

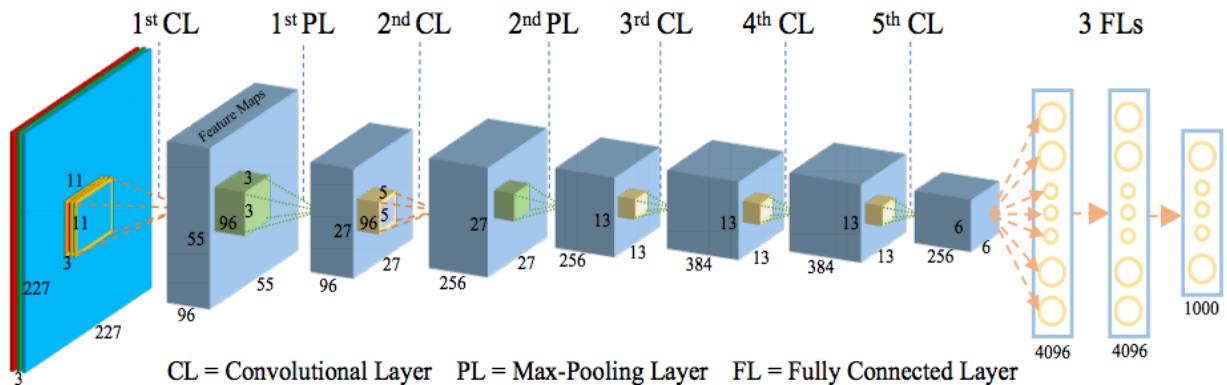


FIGURE 1. *CaffeNet* architecture

Taxonomy of deep CNN architectures showing the seven different categories

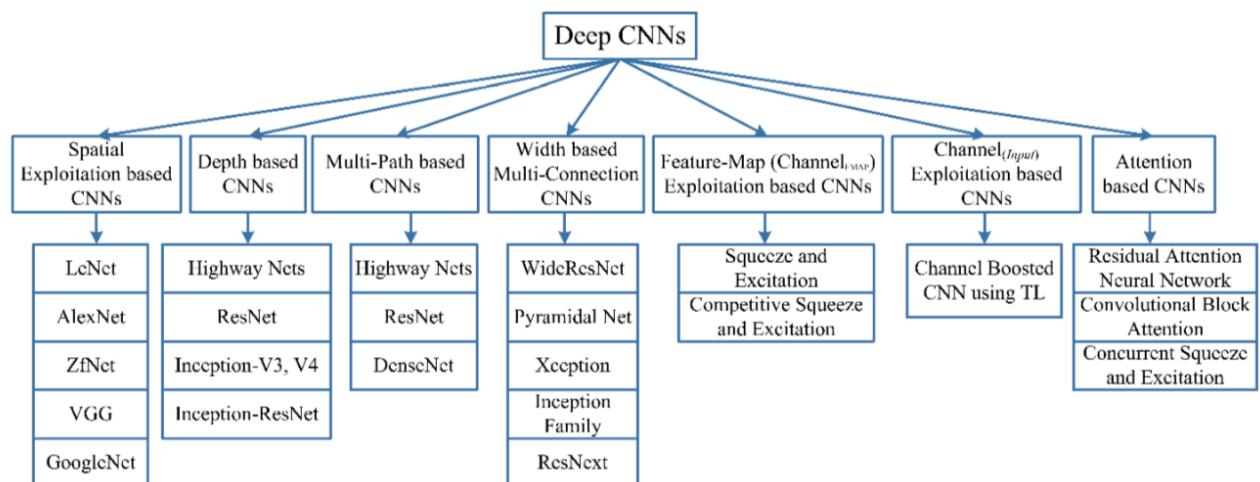


Table 2 Performance comparison of the recent architectures of different categories. Top 5 error rate is reported for all architectures.

Architecture Name	Year	Main contribution	Parameters	Error Rate	Depth	Category	Reference
LeNet	1998	- First popular CNN architecture	0.060 M	[dist]MNIST: 0.8 MNIST: 0.95	5	Spatial Exploitation	[69]
AlexNet	2012	- Deeper and wider than the LeNet - Uses Relu, dropout and overlap Pooling - GPUs NVIDIA GTX 580	60 M	ImageNet: 16.4	8	Spatial Exploitation	[26]
ZINet	2014	- Visualization of intermediate layers	60 M	ImageNet: 11.7	8	Spatial Exploitation	[30]
VGG	2014	- Homogenous topology - Uses small size kernels	138 M	ImageNet: 7.3	19	Spatial Exploitation	[31]
GoogLeNet	2015	- Introduces block concept - Split transform and merge idea	4 M	ImageNet: 6.7	22	Spatial Exploitation	[32]
Inception-V3	2015	- Handles the problem of a representational bottleneck - Replace large size filters with small filters	23.6 M	ImageNet: 3.5 Multi-Crop: 3.58 Single-Crop: 5.6	159	Depth + Width	[126]
Highway Networks	2015	- Introduced an idea of Multi-path	2.3 M	CIFAR-10: 7.76	19	Depth + Multi-Path	[102]
Inception-V4	2016	- Split transform and merge idea - Uses asymmetric filters	35 M	ImageNet: 4.01	70	Depth + Width	[35]
Inception-ResNet	2016	- Uses split transform merge idea and residual links	55.8M	ImageNet: 3.52	572	Depth + Width + Multi-Path	[35]
ResNet	2016	- Residual learning - Identity mapping based skip connections	25.6 M 1.7 M	ImageNet: 3.6 CIFAR-10: 6.43	152 110	Depth + Multi-Path	[33]
DelugeNet	2016	- Allows cross layer information flow in deep networks	20.2 M	CIFAR-10: 3.76 CIFAR-100: 19.02	146	Multi-path	[127]
FractalNet	2016	- Different path lengths are interacting with each other without any residual connection	38.6 M	CIFAR-10: 7.27 CIFAR-10+: 4.60 CIFAR-10++: 4.59 CIFAR-100: 28.20 CIFAR-100+: 22.49 CIFAR100++: 21.49	20 40	Multi-Path	[128]
WideResNet	2016	- Width is increased and depth is decreased	36.5 M	CIFAR-10: 3.89 CIFAR-100: 18.85	28	Width	[36]
Xception	2017	- Depth wise convolution followed by point wise convolution	22.8 M	ImageNet: 0.055	126	Width	[129]
Residual Attention Neural Network	2017	- Introduces attention mechanism	8.6 M	CIFAR-10: 3.90 CIFAR-100: 20.4 ImageNet: 4.8	452	Attention	[41]
ResNeXt	2017	- Cardinality - Homogeneous topology - Grouped convolution	68.1 M	CIFAR-10: 3.58 CIFAR-100: 17.31 ImageNet: 4.4	29 - 101	Width	[34]
Squeeze & Excitation Networks	2017	- Models interdependencies between feature-maps	27.5 M	ImageNet: 2.3	152	Feature-Map Exploitation	[116]
DenseNet	2017	- Cross-layer information flow	25.6 M 25.6 M 15.3 M 15.3 M	CIFAR-10+: 3.46 CIFAR100+:17.18 CIFAR-10: 5.19 CIFAR-100: 19.64	190 190 250 250	Multi-Path	[101]
PolyNet	2017	- Experimented structural diversity - Introduces Poly Inception module - Generalizes residual unit using polynomial compositions	92 M	ImageNet: Single:4.25 Multi:3.45	- -	Width	[38]
PyramidalNet	2017	- Increases width gradually per unit	116.4 M 27.0 M 27.0 M	ImageNet: 4.7 CIFAR-10: 3.48 CIFAR-100: 17.01	200 164 164	Width	[37]
Convolutional Block Attention Module (ResNeXt101 (32x4d) + CBAM)	2018	- Exploits both spatial and feature-map information	48.96 M	ImageNet: 5.59	101	Attention	[40]
Concurrent Spatial & Channel Excitation Mechanism	2018	- Spatial attention - Feature-map attention - Concurrent placement of spatial and channel attention	-	MALC: 0.12 Viscerak 0.09	-	Attention	[117]
Channel Boosted CNN	2018	- Boosting of original channels with additional information rich generated artificial channels	-	-	-	Channel Boosting	[39]
Competitive Squeeze & Excitation Network CMPE-SE-WRN-28	2018	- Residual and identity mappings both are used for rescaling the feature-map	36.92 M 36.90 M	CIFAR-10: 3.58 CIFAR-100: 18.47	152 152	Feature-Map Exploitation	[130]

Table 3 Different online available resources for deep CNNs.

Category	Description	Source
Cloud based Platforms	Online free access to GPU and other deep learning accelerators	Google Colab: https://colab.research.google.com/notebooks/welcome.ipynb
		Colac: https://cocalc.com/
	Commercial platforms offered by world leading companies	FloydHub: https://www.floydhub.com/
		Amazon SageMaker: https://aws.amazon.com/deep-learning/
		Microsoft Azure ML Services: https://azure.microsoft.com/en-gb/services/machine-learning/
		Google Cloud: https://cloud.google.com/deep-learning-vm/
		IBM Watson Studio: https://www.ibm.com/cloud/deep-learning
Deep Learning Libraries	Deep learning libraries that provide built-in classes of NNs, fast numerical computation and automated estimation of gradients both for CPUs and GPUs.	Pytorch: https://pytorch.org/
		Tensorflow: https://www.tensorflow.org/
		MatConvNet: http://www.vlfeat.org/matconvnet/
		Keras: https://keras.io/
		Theano: http://deeplearning.net/software/theano/
		Caffe: https://caffe.berkeleyvision.org/
		Julia: https://julialang.org/
Lecture Series	Online available and freely accessible deep learning courses	Stanford Lecture Series: http://cs231n.stanford.edu/
		Udacity: https://www.udacity.com/course/deep-learning-nanodegree--nd101 https://www.udacity.com/course/deep-learning-pytorch--ud188
		Udemy: https://www.udemy.com/course/deep-learning-learn-cnns/ https://www.udemy.com/course/modern-deep-convolutional-neural-networks/ https://www.udemy.com/course/advanced-computer-vision/ https://www.udemy.com/course/deep-learning-convolutional-neural-networks-theano-tensorflow/ https://www.udemy.com/course深深学习-pytorch/
		Coursera: https://www.coursera.org/learn/convolutional-neural-networks https://www.coursera.org/specializations深深学习
		ImageNet: http://image-net.org/
		COCO: http://cocodataset.org/#home
		Visual Genome: http://visualgenome.org/
Vision Datasets	Online freely accessible datasets of different categories of annotated images	Open Images: https://ai.googleblog.com/2016/09/introducing-open-images-dataset.html
		Places: http://places.csail.mit.edu/index.html
		Youtube-8M: https://research.google.com/youtube8m/index.html
		CelebA: http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html
		CIFAR10: https://www.cs.toronto.edu/~kriz/cifar.html
		Indoor Scene Recognition: http://web.mit.edu/torralba/www/indoor.html
		Computer Vision Datasets: https://computervisiononline.com/datasets
Deep Learning Accelerators	Energy and computation efficient deep learning accelerators	Fashion MNIST: https://research.zalando.com/welcome/mission/research-projects/fashion-mnist/
		NVIDIA: http://nvidia.org/
		FPGA: https://www.intel.com/content/www/us/en/artificial-intelligence/programmable/overview.html
		Eyeriss: http://eyeriss.mit.edu/
		Google's TPU: https://cloud.google.com/tpu/

AlexNet su dataset ImageNet 2012

VGG

Going deeper with convolutions (inception block)

ResNet

La formula per conoscere il numero di nodi MPLN dato l'input (*fan-in* neurone classico) n e l'indirizzamento b . È la seguente:

$$N_{MPLN} = \frac{b^{\log_b(n)+1}-1}{b-1} - n. \text{ (non funziona bene)}$$

n indica un input generico che può essere sia quello del volume, sia quel del *pooling*.

Questa formula è valida per il momento se n è una potenza di b . Negli altri casi bisogna correggere l'errore, cioè esisteranno sempre un numero di output dei *layer* (ovviamente $< b$) che non potrà formare un indirizzo completo per un neurone nello strato successivo.